Express Mail Label
No. EL 849003485 US

Attorney Docket No.
14406US03

APPENDIX C

Title :	Radio Frequency Local Area Network
Inventors:	Meier, et al.
Attorney Docket No.	14406US03

S3

SST NETWORK ARCHITECTURE

54

Network Overview.

The SST system implements a hierarchical radio frequency network of, possibly mobile, terminals used primarily for online data entry and occasionally for batch file transfers. The network is characterized by sporadic data traffic over multiple-hop data paths consisting of RS485 or ethernet wired links and single-channel direct sequenced spread spectrum radio links. The network architecture is complicated by moving nodes, hidden nodes, sleeping nodes, transient radio links, and unidirectional radio links.

The SST network consists of the following types of devices:

Host - A host computer which communicates with terminals in the SST network. A host can be viewed as the network addressable software entity which interfaces the SST network to the physical host computer.

Controller - A gateway which passes messages from a host port to the SST network; and which passes messages from the network to a host port. The host port (directly or indirectly) provides a link between the controller and a host computer to which terminals are logically attached. Each autonomous network must have at least one controller device.

Base station - An interior node which is used to extend the range of a controller node. Base-station-to-controller or base-station- to-base-station links can be wired or radio.

Terminal - Norand hand-held computer, printer, etc. A terminal can be viewed as the network addressable software entity which interfaces the SST network to the physical device.

The devices are, logically, organized as **nodes** in an (optimal) **spanning tree**, with a root node in a controller device, internal nodes in base stations or controllers on branches of the tree, and terminal nodes as (possibly mobile) leaves on the tree. With the exception of the root node, each (child) node is connected by a single logical link to a parent node. Like a sink tree, nodes closer to the root of the spanning tree are said to be "**downstream**" from nodes which are further away. Conversely, all nodes are "**upstream**" from the root. Packets are only sent along branches (i.e. logical links) of the spanning tree. Nodes in the network use a "**backward learning**" technique to route packets along branches of the spanning tree.

Devices in the spanning tree are logically categorized as one of the following three node types:

Root (or root bridge) - A controller device which functions as the **root bridge** of the network spanning tree. The spanning tree has a single root node. Initially, all controllers are **root candidates**. One, and only one, root node is determined for each autonomous network by using a priority-based root selection algorithm.

Bridge - An internal node in the spanning tree which is used to "bridge" terminal nodes together into an interconnected network. Note that the root node is a bridge and the term "bridge" may be used to refer to all non-terminal nodes or all non-terminal nodes except the root, depending on the context. A bridge node consists of a **network interface point** and a **routing function.**

Terminal - a leaf node in the spanning tree. A terminal node can be viewed as the software entity that terminates a branch in the spanning tree. A terminal node consists of a **network interface point** and one or more **terminal access points.**

A **controller device** contains a terminal node(s) and a bridge node. The bridge node is the root node if the controller is functioning as the root bridge.

A **base station** contains a bridge node. A base station does not contain a terminal node.

A **terminal device** contains a terminal node.

Bridging entity refers to a bridge node or to the **network interface point** in a terminal device.

Network interface point refers to the single network addressable entity which must exist in all nodes. A network interface point is equivalent to the software entity which is used to interface the SST network to a device or bridging node. Note that a controller device connected to a host computer will have a network interface point which references the host computer and a second discrete network interface point which references the bridging node in the controller. Each network interface point is identified by a unique network address. Unless otherwise specified, this document uses "network address" or simply "address" to refer to the identifier of a network interface point. Note that multiple network interface points may be referenced with multicast and broadcast addresses.

Terminal access point refers to a higher layer access point into the network. A terminal access point is defined by the concatenation of the network interface point address and the terminal access point identifier. A terminal device or controller device can have multiple terminal access points.

A **logical port** is defined by a physical port and a network interface point. This implies that a single device may have more than one physical port with the same network address. In this document "port" refers to a logical port.

Figure 1.1 illustrates the relationship between devices, nodes, terminal access points (TAP), network interface points (NIP) and network routing functions (NRF). The controller device has two network interface points. As an example, the NIP in the controller's terminal node is equivalent to the software entity which interfaces to a host computer. The two terminal access points attached to that NIP identify discrete applications (i.e. terminal emulation and file transfer applications directed to the host computer).

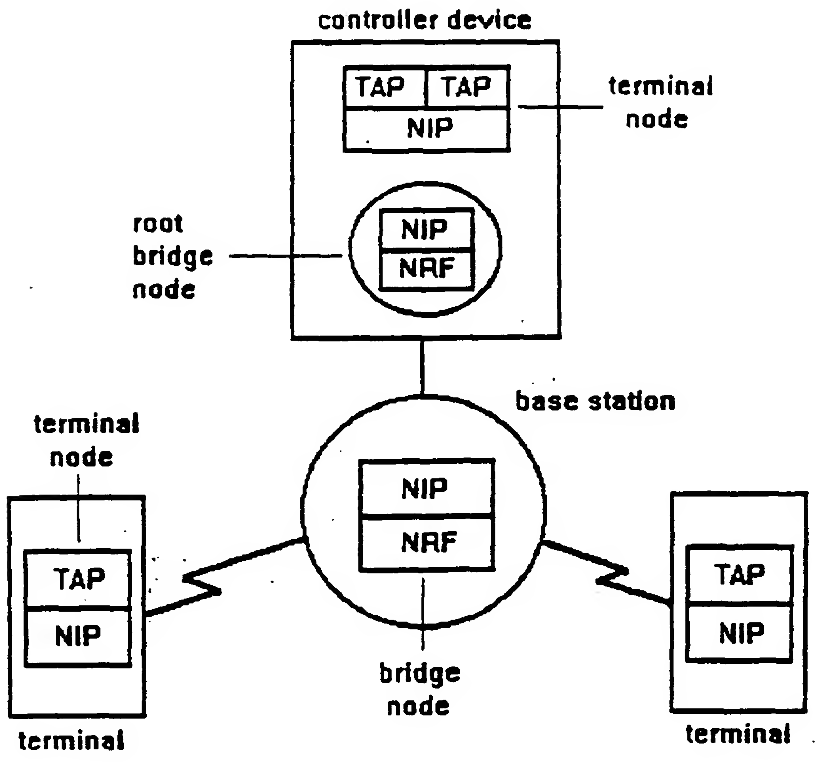

figure 1.1

Basic network requirements:

- Wired or wireless node connections.
- Network layer transparency.
- Dynamic/automatic network routing configuration.
- Terminal mobility. Terminals should be able to move about the radio network without losing a data-link connection.
- Ability to accommodate sleeping terminals.
- Ability to locate terminals quickly.
- Built-in redundancy. Lost nodes should have minimal impact on the network.
- Physical link independence. Higher layer protocols must be consistent across heterogeneous physical links.

Network Layers - Overview.

The SST network software is functionally layered as follows:

Medium Access Control (MAC).

The MAC layer is responsible for providing reliable transmission between ports on any two devices in the network (i.e. terminal-to-base-station). The MAC has a **channel access control** component and a **link control** component. The two components are equivalent to the ISO media access control and data link control sublayers, respectively.

The link control component facilitates reliable point-to-point frame transfers in the absence of collision detection and in the presence of errors.

Several channel access control algorithms are used to regulate access to the communications channel. The algorithms are link-type dependent.

A **p-persistent CSMA/CA** (carrier sense multiple access with collision avoidance) protocol is used to gain access to an **RS485 LAN**. The collision avoidance scheme gives channel access priority to the recipient of a unicast frame.

On lightly loaded spread spectrum radio links, a **non-persistent CSMA** algorithm is used to gain access to the communications channel. Under moderate to heavy channel utilization, an **LBT/BP** (listen-before-talk with busy pulse) algorithm is used to gain access to the channel and minimize the effect of hidden nodes.

A **polling protocol** is used to restrict contention to **request-for-poll (RFP)** frames, thus minimizing contention for data frames.

The channel access control algorithms incorporate a random backoff algorithm to prevent deadlock and instability in contention situations.

Bridging Layer.

The bridging layer has several functions:

1) The bridging layer uses a "HELLO protocol" to organize nodes in the network into an optimal spanning tree rooted at the root bridge. The spanning tree is used to prevent loops in the topology. Interior branches of the spanning tree are relatively stable (i.e. controllers and relay stations do not move often). Terminals, which are leaves on the spanning tree, may become unattached, and must be reattached, frequently.

2) The bridging layer routes packets from terminals to the host, from the host to terminals, and from terminals to terminals along branches of the spanning tree.

3) The bridging layer provides a service for storing packets for SLEEPING terminals. Packets which cannot be delivered immediately can be saved by the bridging entity in a parent node for one or more HELLO times.

4) The bridging layer propagates lost node information throughout the spanning tree.

5) The bridging layer maintains the spanning tree links.

6) The bridging layer distributes network interface addresses.

7) The bridging layer organizes nodes into logical coverage areas on radio channels.

Data-link (transport) layer.

The data-link layer provides an end-to-end data path between data-link access points in any two nodes in the network. The data-link layer provides a connection-oriented reliable service and a connectionless unreliable service. The reliable service detects and discards duplicate packets and retransmits lost packets. The unreliable service provides a datagram facility for upper layer protocols which provide a reliable end-to-end data path.

The data-link layer provides ISO layer 2 services for terminal-to-host application sessions which run on top of an end-to-end terminal-to-host transport protocol. However, the data-link layer provides transport (ISO layer 4) services for sessions contained within the SST network.

Higher Layers.

For terminal-to-terminal sessions contained within the SST network, the data-link layer provides transport layer services and no additional network or transport layer is required. In this case, the MAC, bridging, and data-link layers discussed above can be viewed as a data-link layer, a network layer, and a transport layer, respectively. For terminal-to-host-application sessions, higher ISO layers exist on top of the SST data-link layer and must be implemented in the terminal and host computer, as required. This document does not define (or restrict) those layers. This document does discuss a fast-connect VMTP-like transport protocol which is used for transient internal terminal-to-terminal sessions.

Network address requirements.

MAC frames contain a hop destination and hop source address in the MAC header.

Bridging packets contain an end-to-end destination and source address in the bridging header.

Data-link headers contain source and destination access point identifiers. A data-link connection is defined by the concatenation of the bridging layer source and destination address pairs and the destination and source data-link access points. One end of a connection is equivalent to a terminal access point and is specified as <access_point>@<network_address>, where aliases can be used for both.

MAC and bridging addresses are consistent and have the same format.

All devices must have either a unique **long identifier** which is programmed into the device at the factory and/or an **alias** which is typically entered by the user or is well-known. The long address/alias binds to a short **network address**, obtained from an address server in the root node. A network address uniquely identifies the **netw rk interface point** in each node.

The **network interface point**, in each node, obtains a **network address** from an address server in the root, which uniquely identifies the node. The network interface point passes the network address to the MAC entity attached to each port on a device. Short addresses are used to minimize packet sizes.

Network addresses consist of 2 parts:

- a **node type**. Node types are well-known. Some node types are used for multicast messages (i.e. all bridges).

- a unique, multicast, or broadcast **node identifier**.

A node-type identifier of all 1's is used to specify all node types.

The node identifier part of a root address is all 0's.

A node identifier of all 1's is used to specify any node of the specified type, and is the default node identifier, used before a unique node identifier is obtained.

In addition, to source and destination addresses, each network packet contains a **spanning tree identifier** in the MAC header.

A default spanning tree identifier is well-known by all nodes.

A non-default spanning tree identifier can be entered into the root node (i.e. by a network administrator) and advertised to all other nodes in HELLO.response packets. The list of non-default spanning trees to which other nodes can attach must be entered into each node.

Network address allocation.

The network node identifier of a root node is always all 0's and is well-known. All other nodes must obtain a unique network node identifier from a Reverse Address Resolution Protocol (RARP) server in the root node. A node identifier of all 1's is used until a unique identifier is obtained. To get a unique identifier, a node must send a RARP.request packet to the RARP server. The packet contains the requesting node's unique **long identifier** and/or an **alias** for the long identifier. A **network address** is returned to the requesting node in a RARP.response packet.

Nodes must obtain a (new) network address whenever a new root node is discovered and whenever an ADDRESS_TIMEOUT inactivity period expires without the node receiving a packet from the bridging entity in the root. (A node can prevent its address from expiring by sending an empty **attach.request** packet to the root.)

The address server in the root associates an age factor with each allocated network address. The age factor is incremented each time an ADDRESS_TIMER expires. The age factor, associated with an address, is reset to 0 whenever the root receives a packet from the address. An address is available for use by a requesting node if it has never been used or if it has been inactive for a MAX_ADDRESS_LIFE time period.

MAX_ADDRESS_LIFE must be much larger than ADDRESS_TIMEOUT to ensure that an address is not in use by any node when it becomes available for another node. If the root receives a RARP.request packet from a source for which an entry exists in the address queue, the root simply resets the age factor to zero and returns the old address.

Bridging layer theory and implementation notes.

The bridging layer organizes nodes into an optimal spanning tree with a single root bridge at the root of the tree. (Note that the spanning tree identifier allows more than one logical tree to exist in the same coverage area.) Spanning tree organization is facilitated with a HELLO protocol which selects a root node and enables nodes to determine the shortest path to the root before attaching to the spanning tree. All messages are routed along branches of the spanning tree. Restricting each node to a single parent guarantees that there will be no loops in the logical topology.

Figures 2.1 and 2.2 illustrate how physical devices are organized into logical nodes in a spanning tree. Figure 2.1 depicts devices and the physical communication links. Figure 2.2 depicts the same devices organized as nodes on branches of a spanning tree. The root node in fig. 2.2 is labeled with an R.

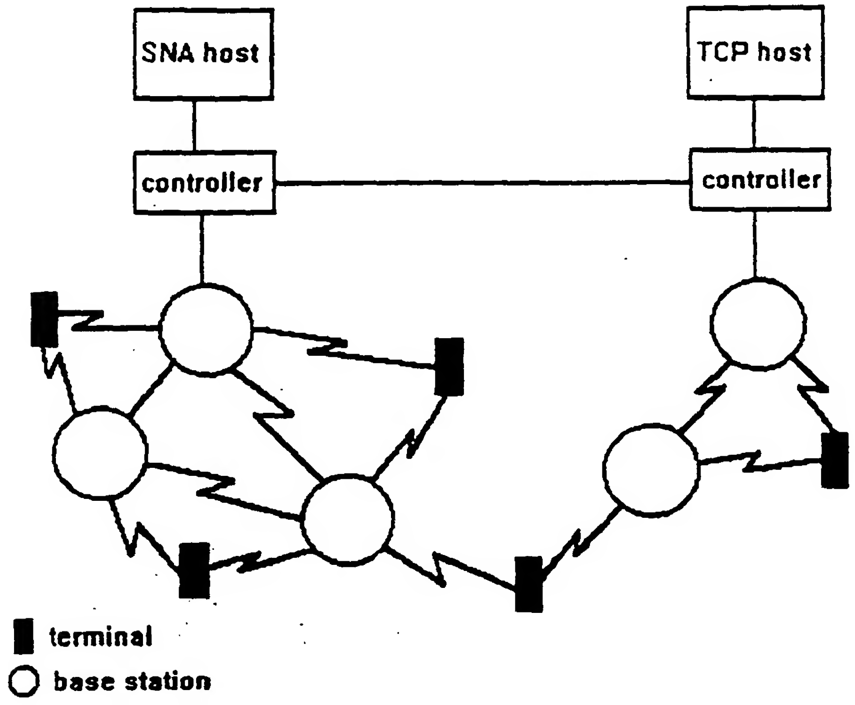

figure 2.1

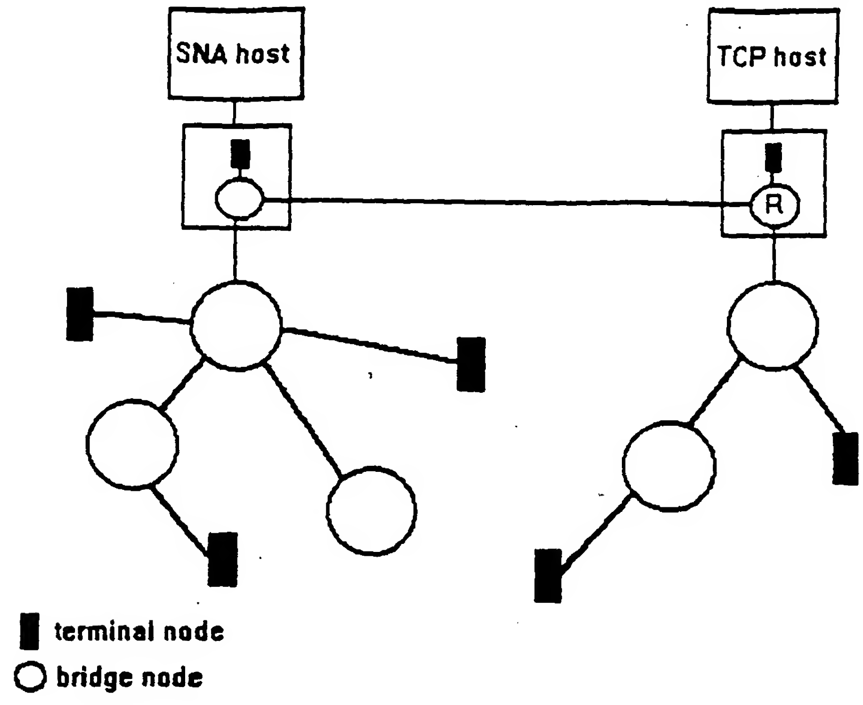

figure 2.2

Spanning tree organization.

Nodes in the network are generally categorized as ATTACHED or UNATTACHED. Initially, only the root is attached. A single controller may be designated as the root, or multiple root candidates (i.e. controllers) may negotiate to determine which node is the root.

Attached bridge nodes and root candidates transmit HELLO.response packets at calculated intervals. The HELLO.response packets include:

- the source address.

- a broadcast destination address.

- the distance (cost) to the root.

The incremental portion of the distance between a node and its parent is primarily a function of the physical link type (i.e. ethernet, RS485, or radio). On radio links, bridging connections are biased toward the link with the best signal strength. Signal strength is not a factor in the cumulative path distance. The distance component is intended to bias path selection toward high-speed (i.e. wired) connections.

- a "seed" value used to calculate the time of the next HELLO.response packet.

- a hello slot displacement. The displacement specifies the displacement of the actual hello slot time from the calculated hello slot time or to indicate that the hello time was not calculated (i.e. was unscheduled).

- a spanning tree identifier (LAN ID).

- the priority of the root node (or root candidate).

- the long, unique device identifier of the root node (or root candidate).

- a root node sequence number, used to distinguish between multiple occurrences of the spanning tree with the same root node.

Attached nodes must forget their network address and return to the UNATTACHED state whenever a HELLO.response packet is received with a new root node identifier or sequence number.

Optional parameters:

- descendent count.

- a pending message list.

Pending message lists consist of 0 or more network addresses for SLEEPING terminals. Pending messages for terminals are stored in the parent node.

- a detached-node list.

Detached-node lists contain the addresses of nodes which have detached from the spanning tree. An internal node learns which entries should be in its list from DETACH packets which are broadcast by internal nodes when a child is lost. Entries are included in HELLO.response packets for DETACH_MSG_LIFE hello times.

Attached nodes broadcast short HELLO.response packets immediately if they receive a HELLO.request packet with a global destination address; otherwise, attached nodes will only broadcast HELLO.response packets at calculated time intervals. Short HELLO.response packets are sent independently of regular HELLO.response packets and do not affect regular hello timing.

Unattached nodes (i.e. without a parent in the spanning tree) are, initially, in an UNATTACHED state. During the unattached state, a node learns which attached bridge is closest to the root node by listening to HELLO.response packets. After the learning period expires an unattached node sends an ATTACH.request packet to the attached bridge node closest to the root. (Nodes without a network address must first send a RARP.request packet to the root to obtain a network address.) The attached node adopts the unattached node as a child by acknowledging the ATTACH.request packet and forwarding it onto the root node. The root node returns the request as an end-to-end ATTACH.response packet. If the newly attached node is a bridge, it calculates its distance to the root, by adding its link distance to the total distance of its new parent, and begins to transmit HELLO.response packets.

The end-to-end ATTACH.request functions as a discovery packet, and enables nodes in the path to the root node to quickly learn the address of the source node.

The UNATTACHED learning state ends after HELLO_RETRY hello time slots if HELLO.response packets have been received from at least one node. If no HELLO.response packets have been received the listening node waits (i.e. sleeps) and retries later.

An attached node may respond to a HELLO.response packet from a node other than its parent (i.e. with an ATTACH.request packet) if the difference in the hop count specified in the HELLO.response packet exceeds a CHANGE_THRESHOLD level.

Unattached nodes may broadcast a global HELLO.request with a multicast bridge destination address to solicit short HELLO.response packets from attached bridges. The net effect is that the UNATTACHED state may (optionally) be shortened. (Note that only attached bridges or the root may respond to an ATTACH.request packet.) Normally, this facility is reserved for terminals with transactions in progress.

ATTACH.request packets may contain a descendants list, so that an internal node may attach itself and the subtree under it to a bridge node closer to the root.

ATTACH.request packets contain a "delivery service type" field, which indicates that a terminal (i.e. which sent the request) may be SLEEPING, and a "maximum stored message count" field. The bridging entity in the parent of a SLEEPING terminal will temporarily store messages, for later delivery, if that service is requested. The bridging entity in a parent node will store pending messages until 1) the message is delivered, or 2) "maximum stored message count" hello times have expired.

Data-link layer data can be piggybacked on an ATTACH.request packet from a terminal.

Attachment criteria.

On wired links, the weighted distance is the only criteria for choosing a parent.

On radio links, a parent is chosen based on the following criteria:

1) The signal strength must exceed a minimum threshold value.
2) If two potential parent nodes are at a different distance from the root, the one with the least distance is chosen.
3) If two potential parent nodes are at the same distance, the node with the best signal strength is chosen.
4) If two potential parent nodes are at the same distance and have the same signal strength, then the node with the highest priority is chosen.
5) If the distance, signal strength and priority are equal, then the node with the highest long ID or alias is chosen.

The intent of the above criteria is to create stable disjoint **logical coverage areas** in the presence of physically overlapping coverage areas. Ideally, all radio terminals in a coverage area will be attached to a single bridge node.

The concept of logical coverage areas is illustrated in the figure below.

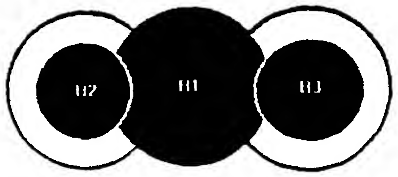

figure 3.1

Three radio bridge nodes, B1, B2, and B3, with network addresses of 1, 2, and 3 are depicted in figure 3.1. All three nodes are assumed to be at the same distance from the root. The circles around each node denote physical coverage areas. The dark shaded areas denote strong signal coverage areas. The logical coverage of B1 is depicted by its dark and light shaded areas. The logical coverage areas of B2 and B3 are depicted by the associated dark shaded and non-shaded areas. Note that the dark shaded area of B1 would logically hide the dark shaded area of B2 or B3 if the areas physically overlapped. Also note that the logical coverage area of a fourth bridge with the sane distance and a higher address, say B4, could be completely hidden behind one of the three bridge nodes, if B4 and one of the other bridges were located side-by-side.

The concept of disjoint logical coverage areas is especially important when radio bridge nodes are placed in close proximity to provide redundant coverage (i.e. for protection against a failure). The MAC entity in one of the bridge nodes can efficiently regulate access to the channel (i.e. by queuing terminals for polling) without coordination with other co-located bridge nodes.

Bridging layer routing.

All packets are routed along branches of the spanning tree. Bridges "learn" the address of terminals by monitoring traffic from terminals (i.e. to the root). When a bridge receives a packet directed toward the root (i.e. moving **downstream**), the bridge creates or updates an entry in its routing table for the terminal. The entry includes the terminal address, and the address of the bridge which sent the packet (i.e. the hop source address). When a bridge receives an **upstream** packet (i.e. moving from the root, destined for a terminal) the packet is forwarded to the upstream node which is specified in the routing entry for the destination. Upstream packets are discarded whenever a routing entry does not exist. Downstream packets (i.e. from a terminal to the root) are simply forwarded to the next downstream node (i.e. the parent) in the branch of the spanning tree. **No explicit routing is required for downstream traffic** since the route is defined by the structure of the spanning tree. A packet travels downstream until a node is reached which has an entry in its routing table for the destination address. The packet is then explicitly routed upstream until it reaches its destination. Thus, **terminal-to-terminal communications is accomplished by routing all traffic through the nearest common ancestor** of both terminals. In the worst case, the root is the nearest common ancestor. An address resolution server in the root node facilitates terminal-to-terminal communications (see below).

For example, in figure 3.1, if terminal E sends a packet to terminal B, the packet will follow downstream hops from E to 3, 3 to 2, 2 to 1, and 1 to R, where R is the root node. Routing tables are not required for the downstream hops. The routing function at R has an entry for B in its routing table which specifies B as the first upstream hop to B. Therefore the packet is explicitly routed upstream from R to B.

As a second example, if terminal D sends a packet to terminal E, the packet will follow one downstream hop from D to 2. The routing function at 2 has an entry for E in its routing table which specifies 3 as the first upstream hop to E. The packet is routed upstream to 3. An entry in the routing table at 3 specifies E as the first upstream hop to E, and the packet is routed from 3 to E.

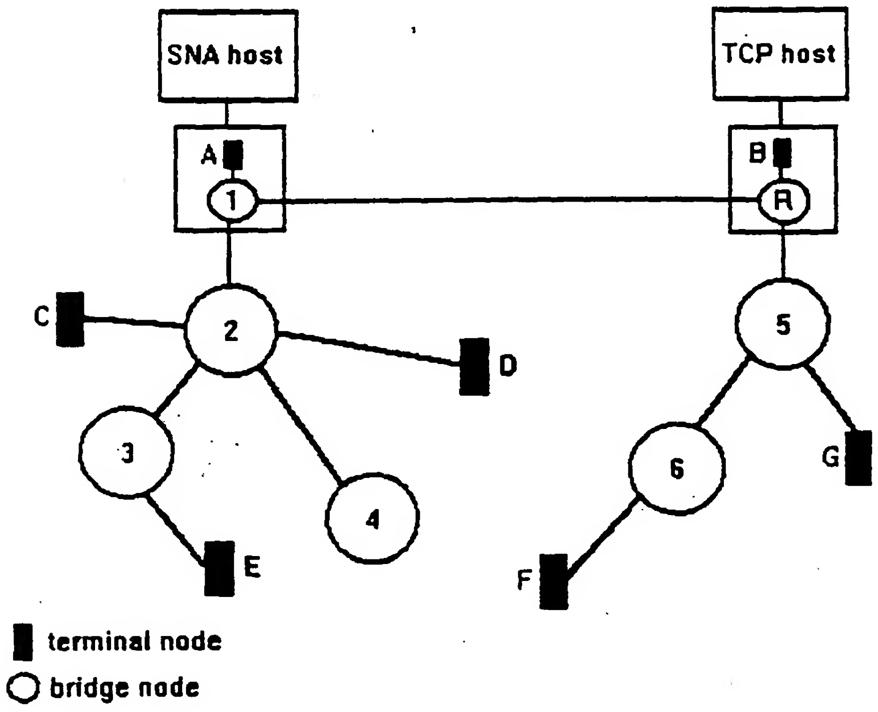

figure 3.4

As an extension to the routing algorithm described above, terminals may optionally cache the addresses of neighbors in a separate direct route table. If a terminal has a message for a destination listed in its direct route table, it may transmit it directly to the source node. Note that the packet may not follow a branch of the spanning tree. Direct route table entries must be aged relatively quickly. If a direct transmission fails, the entry in the direct route table is discarded and the packet is simply forwarded downstream to the root. The header format field in the bridge header must be set to point-to-point for directly transmitted packets. Direct routing has obvious advantages; however it forces terminals to maintain additional MAC layer state information.

As an example of direct routing, in figure 3.4, terminal E can route packets directly to F, if E has an entry for F in its direct routing table.

Note that the direct routing table in a node is built by listening to traffic directed to other nodes. If the MAC layer screens such traffic from the bridging layer, the direct routing table must be built by the MAC layer.

Dynamic Changes in the Spanning Tree.

Paths in the spanning tree can change for a number of reasons:

1) Any node may select a new path to the root whenever a better path is found. For example, a better path might be one where the distance of a node's parent from the root is CHANGE_THRESHOLD greater than the distance in a HELLO.response packet from another node. Note that CHANGE_THRESHOLD can be as small as 1. All else being equal, a node on a radio channel should always choose, as its parent, the node with the best signal strength. A node can move its entire subtree by including a decedents list in the ATTACH.request packet sent to the new parent. Rapidly moving terminals can cache a short list of alternate parents. Periodically, SLEEPING terminals, must stay awake for 1+ HELLO times to discover changes (i.e. shorter paths) in the network topology.

2) A parent node detaches the subtree rooted at a child node, whenever a message cannot be delivered to the child. This occurs when the MAC layer in a parent node fails to deliver a unicast bridging layer packet to a child node. In addition, the bridging entity in a parent node can retain messages for a child terminal node. Terminals request the saved messages by sending a request packet to the parent with an "**awake time**" parameter. If the message is not requested and delivered after "**store message count**" hello times, then the terminal is detached. If the detached node is a bridge node, the parent node will send a DETACH.request packet, to the root node, which contains a decedents list that defines the lost subtree. If the detached child is a terminal, the parent will flood a DETACH.request throughout all branches of the spanning tree using a **reliable flooding mechanism.**

The detached node information which is broadcast in flooded DETACH.request packets is added to the "detached node set" maintained in each bridge node. The detached node set is copied into the detached node list in the bridge's HELLO.response packets. The entries in the detached node list are discarded after MAX_HELLO_LOST+1 HELLO times.

3) A child node goes into the UNATTACHED state whenever its MAC layer fails to deliver a message to its parent. If the child node is a bridge, it must continue to broadcast scheduled HELLO.response packets with an **infinite distance** for MAX_HELLO_LOST+1 times. If the child node is a terminal, it may solicit short HELLO.response packets to shorten the UNATTACHED state. After the UNATTACHED learning state has expired the node reattaches by transmitting an ATTACH.request to the bridge node closest to the root.

4) If a node in an ATTACHED state receives a DETACH packet or a HELLO.response packet with its network address in the detached list, it must enter the UNATTACHED state and reattach to the spanning tree. Note that it may not actually be unattached. The node can shorten the UNATTACHED state by soliciting short HELLO.response packets. After reattaching, the node must remain in a HOLD_DOWN state for MAX_HELLO_LOST+1 hello times. During the HOLD_DOWN state, the node ignores its address in DETACH packet and HELLO.response packet detached lists. After the HOLD_DOWN period expires the node must send a second ATTACH.request to the root, to insure that it is still attached.

5) Entries in routing tables are aged. When routing table space for a new entry is required, either an unused entry or the oldest (i.e. least recently used) entry is selected. If a used entry is selected, then the old information is simply discarded. The aging factor associated with each table entry is reset to 0 each time a new packet from the associated node arrives. A node must periodically send an ATTACH.request packet to the root node to maintain its path in the spanning tree. The ATTACH.request can be piggybacked on a higher-layer data packet if the node is busy. If the root node is on the path to the destination of the data packet then the overhead required to keep the node's address alive is minimal.

6) A SLEEPING node must wake up and enter an ATTACHED listen state whenever a threshold number (i.e. 1 or 2) of HELLO.response packets, from its parent, are missed. The state ends when the node receives a data or HELLO.response packet from its parent. The node enters the UNATTACHED state when a) its address appears in the detached list of a DETACH or HELLO.response packet, or b) a total of MAX_HELLO_LOST consecutive HELLO.response packets are missed.

The time that a node spends in the ATTACHED LISTEN state must be less than the lifetime of detached node information in the network. This insures that a detached node will always enter the UNATTACHED state (i.e. either the node will find its address in a detached node list or the node will miss MAX_HELLO_LOST HELLO.response packets and go into the UNATTACHED state before it sees a "good" HELLO.response packet from its (former) parent.

7) Any node which receives a HELLO.response packet from its parent with an infinite distance immediately enters the UNATTACHED state. If the node is a bridge, it must continue to broadcast HELLO.response packets with an infinite distance for MAX_HELLO_LOST+1 times.

Note that old invalid paths may exist in the spanning tree for a period of time. For example, if a terminal detaches and reattaches to a different branch in the spanning tree, all downstream nodes in the new branch (quickly) "learn" the new path to the terminal. Nodes which were also in the old path change their routing tables and no longer forward packets along the old path. At least one node, the root, must be in both the old and new path. A new path is established as soon as an end-to-end attach request packet from the terminal reaches a node which was also in the old path. Any remaining old path fragment will be disjoint from the new path.

Detach Packet Logic.

A parent node generates a DETACH.request packet whenever it is unable to deliver a message to a child node. The two possible cases are 1) the child is a bridge, and 2) the child is a terminal.

Case 1 - The lost node is a bridge.

When a parent node is unable to deliver a message to a child bridge node, it must send a DETACH.request packet, to the root node, which contains a detached node list that describes the lost subtree. The list will contain all nodes in the routing table of the parent for which the lost bridge was the first upstream hop. All downstream nodes in the path of the DETACH packet must adjust their routing tables by deleting entries which match those in the detached list.

Case 2 - The lost node is a terminal.

Since terminals can be mobile they can be lost often and must be notified quickly. When a parent node is unable to deliver a message to a terminal, it must generate a DETACH.request packet, with the terminal specified in the associated detached node list, and flood the packet throughout all branches of the spanning tree. The packet is forwarded using a reliable broadcast mechanism. The DETACH packet contains a **forward list** which is used to specify which nodes should forward and acknowledge the DETACH.request. Initially, the forward list consists of all bridges which are either children or the parent of the node which generated the packet. Nodes in the forward list acknowledge the DETACH.request, with a DETACH.response, and forward the DETACH.request along all branches of the spanning tree except the branch it was received on, with one exception. A bridge node in the forward list does not forward an entry in the detached list of a DETACH.request if 1) the DETACH.request came from an upstream node, and 2) the upstream node is not the first hop in the routing table entry associated with the entry in the detached list. Upstream bridges, which do not have bridge nodes as children, broadcast the DETACH.request one time (without a forward list).

Note that the destination address used to forward a flooded DETACH.request is global. Therefore, the detached terminal may receive a DETACH.request and quickly learn that it has been detached. All bridge nodes, which receive the DETACH.request, add the detached terminal to their detached node list which is broadcast in HELLO.response packets for MAX_HELLO_LOST+1 times or until the bridge determines the terminal has reattached.

Hello synchronization.

All attached non-terminal nodes broadcast periodic HELLO.response packets at calculated intervals. On the average, the intervals are separated by HELLO_PERIOD seconds. The HELLO.response packet contains a "seed" field used in a well-known randomization algorithm to determine the next hello time for the transmitting node and the next seed. The address of the transmitting node is used as a factor in the algorithm to guarantee randomization. Nodes can execute the algorithm i times to determine the time (and seed) of the i-th hello packet from the transmitter.

After attaching, a bridge chooses a random initial seed and a "hello slot" and broadcasts a hello packet in that slot. The bridge chooses succeeding hello slots by executing the randomization algorithm. If the transmission of a HELLO.response packet is delayed, then the delay is entered into a hello "displacement" field in the packet, so that the calculated time can be accurately derived by a receiver. Cumulative delays are not allowed (i.e. contention delays during the i-th hello transmission do not effect the time of the i+1 hello transmission).

A node initially synchronizes on a HELLO.response packet from its parent. A SLEEPING node can calculate the time of the next expected HELLO.response packet from its parent and can power-down with an active timer interrupt set to wake it just before the HELLO.response packet is transmitted. The bridging entity in a parent node can store a message for a SLEEPING node until the SLEEPING node "requests" the message by notifying its parent that it is awake. A terminal learns that it must request unsolicited saved message by examining the pending message list in the HELLO.response packet. This implementation enables SLEEPING terminals to receive unsolicited messages and relaxes the timing constraints for transaction oriented messages. Retries for pending messages are transmitted in a round-robin order when messages are pending for more than one destination.

Note that a child node, which misses i HELLO.response packets, is able to calculate the time of the i+1 HELLO.response packet.

Data-link/Transport layer theory and implementation notes.

The data-link layer is implemented as an extension of Class 2 Logical Link Control (LLC) as defined in ISO Standard 8802-2.2. The extensions to LLC are 1) an additional unnumbered command frame - SABMX, and 2) 15-bit send and receive sequence numbers. In addition, the implementation must include an adaptive time-out algorithm for retransmissions. Unreliable (type 1) and reliable (type 2) connection-oriented services are provided. The unreliable service is provided for terminals which support a reliable end-to-end transport protocol with a host computer. LLC type 2 provides a reliable end-to-end transport service for long-lived terminal-to-terminal connections within the spanning tree network.

In addition, a fast-connect VMTP-like transport protocol, is used for transient terminal-to-terminal connections. The VMTP-like service is primarily provided for remote procedure calls (RPC), client/server transactions, and short mail messages.

Note that the bridging layer does not provide a reliable end-to-end service and that lost and duplicate packets must be handled by a higher layer. The bridging layer does not fragment packets and packets are normally delivered in sequence.

The interfaces to the next upper (i.e. application) layer include:

```
handle=CONNECT(destination, . . .)

handle=LISTEN(SSAP, . . .)

SEND(handle, buffer, length, [destination])

DATAGRAM(handle, buffer, length, [destination]);

TRANSACTION(handle, tx_buf, tx_len, rx_buf, max_rx_len,
                    IDEMPOTENT, destination)

RECEIVE(handle, buffer, max_length, [destination])

PENDING_MESSAGE(handle, [destination])

DISCONNECT(handle)
```

Destination fields are formed by concatenating the destination service access point (DSAP) with the destination network address, where aliases are used for both . For example, 3270@HOST1 might designate a 3270 terminal controller application in a controller node. The DSAPs for common applications are well-known. Note that the DSAP can specify a remote terminal application or the access point to a higher layer protocol in a remote node.

The "handle" designates the connection type, and is the connection identifier for LLC connections.

The optional "destination" field in send and receive operations is only used for the VMTP-like interface.

SEND messages require a response.

DATAGRAM messages do not require a response. DATAGRAM is used to send messages to a host which is capable of supporting end-to-end host-to-terminal transport connections.

TRANSACTION is used to send transaction-oriented messages with the VMTP-like facility. An error occurs if a return message is not received in a TRXN_TIME-OUT period. The data-link/transport entity saves response messages and resends the response when a duplicate transaction message is received. In addition, an application can mark a transaction as redoable, by setting the IDEMPOTENT flag ON. In this case, the response message is not saved and the response is regenerated by re-executing the transaction. Note that a response message can be guaranteed in the form of an acknowledgment from a higher layer protocol.

Because the bridging layer provides an unreliable service, the data-link layer is required to detect duplicate packets and retransmit lost packets. Detecting duplicates is facilitated by numbering data-link packets with unambiguous sequence numbers.

Data-link connections.

LLC type 2 connections are established by sending a SABMX control frame to the destination network address. To prevent frames from an old connection from being accepted (i.e. with a sequence number of 0) the node which initiates a connection must insure that at least MAX_PACKET_LIFE time has expired since the last connection before issuing a new CONNECT for the same destination. Because of the required waiting period. Type 2 LLC connections are not ideal for the type of transient "connections" needed to reliably facilitate remote procedure calls, client/server transactions, and sporadic mail messages.

LLC frames are sequenced from 0 to MAX_SEQ. The maximum number of outstanding frames (i.e. transmitted, but not acknowledged) is LLC_WINDOW_SIZE. The default value of LLC_WINDOW_SIZE is relatively small, but the window size may be expanded with an XID frame. Since all frames sent during a connection may not follow the same path, no more than MAX_SEQ frames may be sent in a MAX_PACKET_LIFE time period. A problem can arise when a node successfully transmits a data-link frame to the next downstream hop on a busy path, but loses all acknowledgments. At this point, the node is detached and must quickly reattach to the spanning tree. If the next parent of the node is on a shorter, less busy branch, frames on the new path can easily arrive at the destination while old frames still exist in the old path. MAX_PACKET_LIFE is equal to (MAX_HOPS · XMIT_Q_SIZE · MAX_RETRY_TIME), where MAX_HOPS is the maximum length of a branch of the spanning tree, in hops, XMIT_Q_SIZE is the number of packets which can be queued in each node, and MAX_RETRY_TIME is the maximum time the MAC layer can spend retrying a frame before it is successfully sent. This problem was addressed by increasing the size of the send and receive sequence number fields (i.e. from 7 bits to 15 bits) so that the N(S) and N(R) fields in an information frame can never roll over faster than MAX_PACKET_LIFE time. Note that the spanning tree topology insures that packets will not loop.

VMTP-like connection records are built automatically. A VMTP-like connection record is built (or updated) whenever a VMTP-like transport message is received. The advantage is that an explicit connection request is not required. A VMTP-like connection is half-duplex. (A full-duplex connection at a higher layer can be built with two independent half-duplex VMTP-like connections.) Acknowledgments must be handled by higher layers.

Connections are defined by the concatenated network end-to-end destination and source addresses, and destination and source service access points.

The LLC type 2 data-link entity in a node stores messages for possible retransmission. Note that retransmissions may not always follow the same path (primarily) due to moving terminals and the resulting changes in the spanning tree. For example, the bridging entity in a parent node may disconnect a child after the MAC entity reports a message delivery failure. The child will soon discover that it is detached and will reattach to the spanning tree. Now when the data-link entity (i.e. in the root) resends the message, it will follow the new path.

Data-link message timing and sleeping terminals.

The data-link entity in a terminal calculates a separate time-out for SEND and TRANSACTION operations. Initially, both time-outs are a function of the distance of the terminal from the root node.

A TCP-like algorithm is used to adjust the expected propagation delay for each message type to the end-to-end distance and load, without causing sporadic changes or dramatic swings in time-out values. Messages, which require a response, are retransmitted if twice the expected propagation time expires before a response is received. SLEEPING terminals can power down for a large percentage of the expected propagation delay before waking up to receive the response message. Note that missed messages may be stored by the bridging entity in a parent node for "store message count" hello times.

The LLC entity at each end of a connection should be notified when a bridging layer path change has occurred. The notification will trigger an RR or data LLC message transmission if unacknowledged messages exist on the connection, or an expected reply has not been received.

Medium Access Control (MAC) theory and implementation notes.

The MAC layer is responsible for providing reliable transmission between any two nodes in the network (i.e. terminal-to-bridge).

Access to the network communications channel is regulated in several ways:

- Nodes are grouped into logical coverage areas associated with a single bridge node.
- CSMA and LBT algorithms are used to gain access to the channel.
- A polling protocol reduces contention for data frames.

IEEE 802.3 media access control is used for ethernet links.

A p-persistent CSMA/CA with ARQ (automatic retry request) protocol is used to gain access to the channel on the RS485 LAN.

A **collision avoidance protocol** is implemented on RS485 LAN links. Bridging layer packets are typically sent in a single MAC layer data frame on both ethernet and RS485 LAN links. Short blocks can be transmitted as soon as an idle channel is detected. Before a long data frame can be transmitted on a wired link a potential transmitter must sense an idle channel, transmit an RFP frame and receive a POLL frame from the receiver. After a data frame is transmitted, the receiver notifies all listening nodes that the channel is free by sending a CLEAR frame.

A simple **return priority mechanism** is implemented by requiring a potential transmitter to sense an idle line for an IDLE_TIME period which exceeds the maximum transmitter/receiver turnaround time. The recipient of a unicast frame "owns" the channel for the turnaround time and can respond without executing the CSMA algorithm. This approach makes response times more deterministic and allows the sender to set response time-outs tightly. Short time-outs allow transmitting nodes to quickly retry out and discover disconnected links.

A **CSMA random backoff algorithm** specifies backoff delays as a function of the CSMA slot time. A CSMA slot is calculated as a function of the worst-case carrier sense ambiguous period. If, for example, in the worst case, it takes a character-time to determine that a frame is in progress then the CSMA slot time is defined to be slightly longer than one character time. The algorithm divides the sense time into p contiguous slots and chooses a number, i, between 1 and p. If the first i slots are idle then the algorithm allows transmission in the i+1 slot. If one of the first i slots is busy, then the device executing the algorithm listens until the channel is idle and re-executes the algorithm.

A **polling protocol**, which is consistent with the collision avoidance protocol used on wired links, is used to gain access to the channel on spread spectrum radio links. The polling protocol reduces contention, in an environment with **hidden terminals**, in several ways.

On radio links, a MAC transmitter fragments a bridging layer packet into short fixed length frames before the packet is sent. The fragments are reassembled by the receiver and are posted to the receiver's bridging layer if, and only if, all frames in the packet are received. A group of frames which is associated with a single bridging layer packet is called a **bracket**. Fragmentation at the MAC layer allows the MAC entity to used a (shorter) frame size which is suitable for the link error rate without impacting packet sizes at the bridge layer.

On radio links, the polling protocol generally limits contention to RFP frames. Before a bracket of frames can be transmitted on a radio link, a potential transmitter must sense an idle channel, transmit an RFP frame and receive a POLL frame from the receiver. If the receiver is busy it will respond with a wait-for-poll (WFP) frame. The WFP frame positively acknowledges the RFP frame and causes the transmitter to wait for a POLL frame. Nodes are queued for polling in the order in which RFP frames arrive. After the last frame in a bracket is transmitted and successfully received, the receiver sends a CLEAR frame to notify all listening nodes. Since coverage areas tend to be logically disjoint a single bridge can generally determine access to the channel, within its coverage area, for the transmission of all frames except for RFP (and ENQ) frames directed to the bridge.

The MAC entity attached to a radio port must monitor traffic volume on the port. Under lightly loaded conditions a non-persistent **CSMA/CA with ARQ** algorithm is used to gain access to the channel. A node simply listens and transmits an RFP (request-for-poll) frame immediately if the channel is sensed to be idle. If the channel is busy, the node defers until later.

Under moderate to heavy load conditions, the polling protocol incorporates a p-persistent **LBT/BP with ARQ** algorithm to regulate access to the channel. As Kleinrock and Tobagi noted in [1], the throughput performance of a packet radio network can be significantly degraded by as few as 5 to 10 per cent hidden nodes. In [3] Tobagi discussed the use of a busy tone on a separate frequency. The disadvantage of the busy tone solution is that it requires two transmitters and receivers per node. The LBT/BP protocol provides a somewhat equivalent solution with the use of a single frequency. After the first RFP frame, all other frames sent in a sequence of frames associated with a single bracket are sent without sensing the channel. Because of this **return priority mechanism** and because the MAC entity fragments packets into frames of a fixed size (or smaller) the time between POLL frames (and DATA frames) is fixed. The POLL (or DATA) frames in a bracket provide a **busy pulse** which notifies other nodes in the coverage area of either the transmitter or receiver that a conversation is in progress. All nodes are required to sense an idle channel for a time which exceeds the interpoll gap time before transmitting an RFP frame. The LBT/BP protocol insures that an active conversation will be detected if either the active transmitter or receiver are in range (but not necessarily both). Figure 4.1 below further illustrates this point.

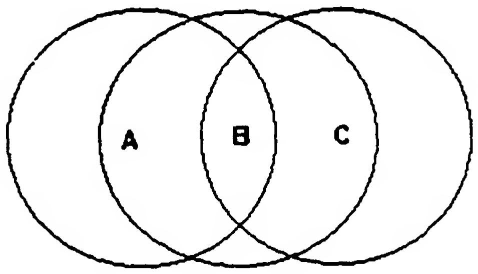

figure 4.1

The example in figure 4.1 depicts a three nodes, A, B, and C, with physical radio coverage areas represented by the circle around each node. Note that B is in the coverage area of both A and C, A is in the coverage area of B but not in the coverage area of C, and C is in the coverage area of B but not in the coverage area of A. Assume that B is polling A to solicit data frames when C first attempts to acquire the channel. Since the LBT algorithm requires C to detect an idle channel for a time which exceeds the interpoll gap time. C will detect a poll frame from A and will defer from transmitting until the conversation is finished.

An LBT **random backoff algorithm** for radio links specifies backoff delays as a function of the CSMA slot time and the **LBT slot** time. An LBT slot is defined as a function of the worst case busy-sense time. In figure 4.1, suppose nodes A and C attempt to initiate a conversion with B at approximately the same time. If A determines that the channel is idle and begins transmitting an RFP frame at time 0, then the worst case busy-sense time is equal to the time, t, at which C begins sending a poll frame.

The backoff algorithm is executed, whenever a collision is suspected, to randomly distribute retries over an increasing number of LBT slots. The channel access algorithm used on radio channels treats "busy sense" as a collision and assumes that a collision may have occurred whenever a response is missed, since collision detection is not available.

The bridge layer is responsible for indicating to the MAC layer that a packet is being sent as a response to a multicast or broadcast message. If a packet is being sent in response to a multicast message, then the MAC layer waits for a random delay period before transmitting the response.

Broadcast and multicast frames are never retransmitted at the MAC layer.

The retry time of the MAC must be relatively short so that lost nodes can be detected quickly. When the MAC layer reports a failure to deliver a message to the bridging layer, the bridging layer can 1) save messages for SLEEPING terminals for later attempts, or 2) DETACH the node from the spanning tree.

The node identifier part of the MAC address is initially all 1's for all nodes except the root node. The all 1's broadcast address is used to by a node to send and receive address resolution packets, which are used to obtain a network address.

Address resolution.

Reverse Address Resolution Protocol (RARP) protocol overview.

A node which does not yet have a 16-bit network address must request a network address from a RARP server in the root node. The node uses an 11-bit node ID of all 1's until a unique ID is assigned by the RARP server. A RARP.request packet, containing the requesting node's unique 48-bit long ID and/or an alias (i.e. LPT1) for the 48-bit identifier, is sent to the server by the requesting node. The RARP server routes a RARP.response packet back to the requesting node using temporary RARP routing tables. A table entry is created in each node on the path to the root when the request is sent. When the RARP.response packet returns upstream, each node records the network address assigned to the requesting node. On the last hop, the RARP.response packet is broadcast to the requesting node. If the packet is not delivered successfully, a retry is initiated by the bridging entity in the requesting node after a BRIDGE_TIMEOUT period expires. The parent of the requesting node can service the retry without forwarding the retry RARP.request to the root. In general, if a bridge node receives a RARP.request and the long identifiers in the request matches an entry in its RARP routing table, and that entry has a valid 16-bit network address, then the bridge node can simply return a RARP.response, with the 16-bit address, without forwarding the request to the root node. Note that RARP routing table entries are aged and discarded before the address can become invalid.

A requesting node can specify that it does not have a 48-bit ID by not including it as a parameter or by including an ID of all 0's. An alias is required if there is no 48-bit ID.

A requesting node can specify that it does not have an alias by not including it as a parameter, or by including an alias with a length of 0.

The long identifiers in a RARP.request match temporary entries in a RARP routing table 1) if matching 48-bit IDs and aliases are present in both, 2) if both contain a matching alias and neither has a 48-bit ID, or 3) if both contain a matching 48-bit ID and neither has an alias.

Note that the RARP request function serves two purposes: 1) A 16-bit network address is obtained, and 2) The long ID and/or alias of the requesting node are registered with the address server.

Address Resolution Protocol (ARP) protocol overview.

A node can request the 16-bit network address of another node by sending an ARP.request packet to an ARP server in the root node. The ARP.request packet must contain either the 48-bit identifier or the alias of the target node. The ARP server returns the 16-bit network address of the target node in an ARP.response packet.

Network Management.

An SNMP-like network management provided is provided through a dedicated network control point which exists in a controller device. A network manager can retrieve and manipulate supported variables in devices distributed throughout the network to control devices and diagnose problems.

Boot Protocol.

A system load protocol similar to IEEE 802.1E is implemented by maintaining a load file database in a controller. Devices can request load files by class or by name. Alternatively, a network manager can distribute load files to devices over the network and can cause devices to reboot with the SNMP-like network management protocol described above.

References.

[1] B. M. Leiner, D. L. Nielson, and F. A. Tobagi, "Issues in Packet Radio Design", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[2] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part I - Carrier Sense Multiple Access Modes and the Throughput Delay Characteristics", IEEE Transactions on Communications, Vol. COM-23, No. 12, December 1975.

[3] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part II - The Hidden Terminal Problem in Carrier Sense Multiple Access and the Busy Tone Solution", IEEE Transactions on Communications, Vol. COM-23, No. 12, December 1975.

[4] L. Kleinrock and F. A. Tobagi, "Packet Switching in Radio Channels: Part IV - Stability Considerations and Dynamic Control in Carrier Sense Multiple Access", IEEE Transactions on Communications, Vol. COM-25, No. 10, October 1977.

[5] L. Kleinrock and J. Silvester, "On the Behavior of Multi-hop Packet Radio Networks", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[6] M. B. Pursley, "The Role of Spread Spectrum in Packet Radio Networks", Proceedings of the IEEE, Vol. 75, No. 1, January 1987.

[7] J. O. Onunga and R. W. Donaldson, "Performance Analysis of CSMA with Priority Acknowledgments (CSMA/PA) on Noisy Data Networks with Finite User Population", IEEE Transactions on Communications, Vol. 39, No. 7, July 1991.

[8] L. Kleinrock and J. Silvester, "Spatial Reuse in Multihop Packet Radio Networks", Proceedings of the IEEE; Vol. 75, No.1, January 1987.

[9] F. Backes, "Transparent Bridges for Interconnection of IEEE 802 LANs", IEEE Network, Vol. 2, No. 1, January 1988.

[10] International Standard ISO/DIS 8802-2.2.

[11] A. S. Tanenbaum, "Computer Networks", Prentice Hall, Second Edition

[12] D. E. Comer, "Internetworking with TCP/IP", Prentice Hall

[13] D. Cherition, "VMTP: Versatile Message Transaction Protocol, protocol specification", TCP/IP Network Working Group, RFC 1045

[14] IEEE standard 802.1E-1990, "Local and Metropolitan Area Networks, System Load Protocol"

[15] M. T. Rose, "The Simple Book"